A NOVEL APPROACH FOR ROBUST WEB SERVICES PROVISIONING

Quan Z. Sheng
School of Computer Science, The University of Adelaide, Adelaide, SA 5005, Australia

Anne H. H. Ngu
Department of Computer Science, Texas State University, San Marcos, TX, USA

Keywords: Web services, mobile agent, computing resource, matchmaking, resource planning.

Abstract: Availability and reliability of Web services are important issues for developing many electronic business applications. Unfortunately, it is hard to guarantee the availability of a service given that the number of its requests might be potentially huge. In this paper, we propose a novel approach for robust Web service provisioning based on mobile agent and resource discovery technologies. With our approach, new service instance can be instantiated at appropriate idle computing resources on demand, therefore reducing the risk of service being unavailable. We present a matchmaking algorithm for resources selection, as well as a multi-phase resource planning algorithm for composite Web services.

1 INTRODUCTION

Web services, and more in general Service-Oriented Architectures (SOAs), are gathering considerable momentum as the technologies of choice to implement distributed systems and perform application integration (Alonso et al., 2004). Recently, more and more enterprises are encapsulating their applications as Web services so that service consumers (either end users or applications) are able to locate and invoke those applications.

Some important issues for Web service provisioning that have deterred its wide adoption are *responsiveness* and *robustness* (Curbera et al., 2003; Ingham et al., 2000; Keidl et al., 2003). It is costly for an enterprise if its services are down, for even a few minutes. Situations like service delay and unavailability could be vital and are not acceptable for many important Web applications (e.g., mission-critical applications of companies). Unfortunately, given that the number of requests of a Web service can potentially be large, a single service host may not be sufficient to provide quick response time and high availability. It is advantageous for a service to be initialized at multiple hosts and for the invocation request of the service to be always directed to the host with the lowest workload.

Code mobility is a powerful concept in general for handling load balancing, performance optimization, fault tolerance, and network disconnections (Fuggetta et al., 1998). Code mobility can contribute significantly to prompt service responses in the sense that heavily loaded or unavailable service hosts can be replaced dynamically by lightly loaded service hosts. Many available mobile code technologies like Aglets, Java, and Sumatra (Acharya et al., 1997) can provide the foundation for the development of novel solutions for robust Web services provisioning.

In this paper, we propose an approach for robust Web services provisioning that builds upon our earlier work in (Benatallah et al., 2003). The core part of our design is a service container component called *service migrator*. A service migrator, which is associated with a Web service, can instantiate a new copy of the service at an idle computing resources on demand or during runtime and therefore, reduces the risk of the service being unavailable when the number of requests to a specific service are becoming very large. Furthermore, with load balancing, faster processing of service requests can be achieved. To facilitate dynamic resource selection for migration, we introduce a *matchmaking* algorithm for selecting computing resources that meet the requirements of Web services. To achieve optimized overall performance of a com-

Z. Sheng Q. and H. H. Ngu A. (2007).
A NOVEL APPROACH FOR ROBUST WEB SERVICES PROVISIONING.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - SAIC*, pages 121-128
DOI: 10.5220/0002357901210128

posite Web service, we also propose a multi-phase resource planning approach where resources are selected for the components of the composite service based on a number of criteria such as communication cost, or availability.

The paper is organized as follows. In Section 2, we introduce our model for robust services provisioning, in particular the functionalities of service migrators and their interactions with other components such as service controllers, matchmaker, and computing resources. Then, in Section 3, we present our algorithm for resource matchmaking. In Section 4, we propose a multi-phase resource planning algorithm for composite Web services. Finally, in Section 5, we discuss the related work and outline future research directions.

2 THE MODEL FOR ROBUST SERVICE PROVISIONING

In this section, we first introduce the concept of mobile Web services and followed by the description of our model for robust services provisioning.

2.1 Mobile Web Services

We distinguish between *stationary* and *mobile* services. Stationary services are location dependent. Such a service cannot move because e.g., the service needs to access a DBMS that is only available on its dedicated server. Meanwhile, mobile Web services are services with migration ability (Keidl et al., 2003). They are similar to mobile agents which have the ability to interact locally with a service or select a specific computing resource to use. In addition, mobile Web services can function just like stationary Web services. They can dynamically interact with other services or clients (e.g., accepting invocation requests, form part of a composite service). Mobile Web services are location independent. They are stateless (i.e., the internal state of such a service is discarded after a request is processed) and do not require special resources or permissions. A mobile Web service can therefore be executed on arbitrary service hosts whose capabilities satisfy the requirements of the service.

Mobile Web services provide a number of distinct features. Firstly, mobile Web services can migrate to the client sites where services are invoked as local calls. It is extremely useful when the data (input and output of the services) are huge since the data do not have to be streamed over on the network. Secondly, mobile Web services can dynamically select a host to migrate so that they can use the selected computing resources to accomplish their tasks or to achieve load balancing. Thirdly, mobile Web services have provided a solution to the robust service provisioning in mobile computing environment in which limited resources and unstable connections are a norm. Services can always be migrated to more stable hosts.

Mobile Web services have recently attracted a significant interest (Chen and Petrie, 2003; Liu and Lewis, 2005; Ishikawa et al., 2004; Keidl et al., 2003). The work proposed in (Ishikawa et al., 2004) considers mobile Web services as synthesis of Web services and mobile agents. While the work in (Liu and Lewis, 2005) develops an XML-based mobile code language called X# and presents an approach for enabling Web services containers to accept and run mobile codes. It should be noted that proposing tools for the development of mobile Web services is not the focus of our work. Instead, we use the concept of mobile Web services in our model as the basis for the robust services provisioning.

2.2 System Design

Central to our design toward a robust provisioning of Web services are a set of *service hosts*, a *monitoring service*, and a service container that consists of two components, namely *execution controller* and *service migrator*. A service host is a computing resource that is running an engine where service codes can be executed. Service hosts can register themselves at a UDDI registry using appropriate tModel[1] (e.g., `ServiceHost`) so that they can be found by service requesters. A monitoring service monitors the status of a computing resource or the server of a Web service (e.g., CPU and memory usage). The execution controller interacts with the monitoring services and coordinates the execution of the corresponding Web service. If the value of a monitored item is beyond a threshold —which can be set by the service provider—and the service is mobile, the controller sends a migration request to the service migrator, which moves the execution of the service to another service host. If no such a service host is available or the Web service cannot move, the controller triggers an exception handling policy, if such a policy has been specified by the service provider.

A service migrator is a light weight scheduler that helps the controller to execute the associated mobile Web service in other computing resources whenever it is necessary. In particular, the service migrator is responsible for:

[1]In UDDI, a tModel provides a semantic classification of the functionality of a service or a resource, together with a formal description of its interfaces.

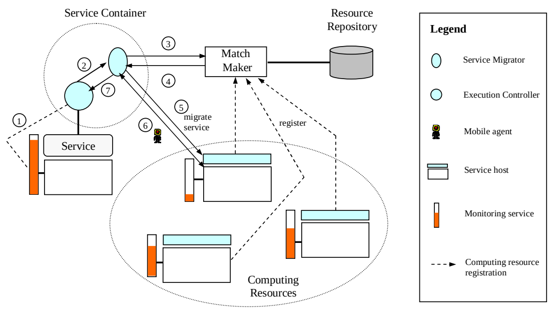


Figure 1: Interactions of system components.

- Receiving service migration requests from the execution controller and sending a matchmaking request to the *matchmaker* to find an appropriate computing resource. The matchmaking request includes the relevant information (e.g., ability to book airline tickets) that will be used by the matchmaker for the resource selection.
- Loading the service codes (of mobile Web services) onto the computing resource recommended by the matchmaker,
- Dispatching a mobile agent to the site of the resource for invoking the service. Upon completion of the invocation, the results are carried back by the mobile agent to the service migrator, which in turn, notifies the service controller about the completion of the invocation.

Figure 1 gives an overview of the interactions among different components during the service migration. When a monitored item reaches a particular threshold (e.g., the CPU usage of the server is higher than 90%, see step 1), the controller of a service sends a migration request to the service migrator (step 2). The migrator interacts with the matchmaker for the recommendation of an appropriate service host (steps 3 and 4). If such a service host exists, the migrator loads the service code to the host and dispatches a mobile agent for the service invocation and result collection (steps 5 and 6). Otherwise, the migrator informs the controller, which then may trigger exception handling policies (e.g., forward the service invocation to an alternative Web service).

The selection of computing resources is based on a matchmaking algorithm that is implemented in the matchmaker. We will give a detailed description of the resource matchmaking in Section 3.

The overall performance of a composite Web service, which is an aggregation of several Web services to fulfil a complex task (e.g., travel planning), can be improved by code mobility (Casati and Shan, 2001). For example, multiple component services in a composite service could be gathered in a single site or several sites in the same domain for the invocation to reduce the communication cost. In Section 4, we will present an execution planning approach to optimally select resources on which the execution of the component services take place.

3 RESOURCE MATCHMAKING

The resource matchmaking consists of two main steps: 1) service migrators and resource providers advertise their requirements and characteristics of the Web services and resources; a designated matchmaking service (i.e., *matchmaker*) matches the advertisements in a manner that meets the requirements and constraints specified in the respective advertisements.

The descriptions of Web services and resources consist of two parts: *attributes* and *constraints*. The attributes part includes characteristics of a service (e.g., service location, mobility, input and output parameters) or a resource (e.g., CPU usage, free memory, price). While the constraints part includes constraint expressions defined by a service or a resource provider, indicating the requirements to select or allocate resources. For example, the provider of a resource may specify that the resource will not be offered to any request from company A because e.g., it always delays the payments. Similarly, the provider of a (mobile) Web service may specify that only resources with more than 200K bytes of free disk space and at least 128K bytes of free memory are eligible to invoke the service.

Resources can be described using W3C's RDF (Resource Description Framework)[2], while Web services can be described using WSDL (Web Service Description Language)[3]. We will not give detailed description of RDF and WSDL due to space limitations. In the rest of this section, we will focus on the introduction of our approach on resources matchmaking and selection.

3.1 Matchmaking Process

Matchmaking is defined as a process that requires a service description as input and returns a set of *matched* computing resources. A computing resource is matched with a service if both the requirement expressions of the service and the constraints of the resource are evaluated to true.

[2]http://www.w3.org/TR/REC-rdf-syntax.
[3]http://www.w3.org/TR/wsdl.

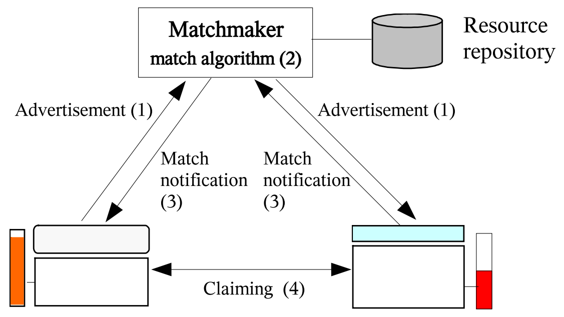


Figure 2: Interactions in the matchmaking process.

Matchmaking Interactions. Figure 2 shows the interactions in a matchmaking process. Service migrators and resource providers construct advertisements describing their requirements and attributes and send them to the matchmaker (step 1). The matchmaker then invokes a *matchmaking algorithm* by which matches are identified (step 2). The invocation includes finding service-resource description pairs that satisfy the constraints and requirements of resources and services. We will describe this step in more detail later. After the matching step, the matchmaker notifies the service migrator and the resource providers (step 3). The service migrator and the resource provider(s) then contact each other and establish a working relationship (step 4). It should be noted that a matched resource of a service does not mean that the resource is allocated to the service. Rather, the matching is a mutual introduction between services and resources and the real working relationship can be consequently built after the successful initial communication between the two partners.

The separation of matching and claiming has some significant advantages. The most important benefit from the separation is the resilience to changes of resources and services. As we mentioned before, the state of Web services and resources may be changing continuously in dynamic environments. There is a possibility that the matches made by the matchmaker are based on out-dated advertised information. Claiming allows the provider of services and resources to *verify* their requirements and constraints in terms of their current states. In addition, a more secure system has resulted from the separation in the sense that in the claiming phase, the providers of resources and services could verify their identities using cryptographic techniques before building the working relationship.

Expression Evaluation. Expression evaluation plays an important role in the matchmaking process. To evaluate a constraint expression in a advertisement (service description), the attribute of the expression is replaced with the value of the corresponding attribute of the resource. If the corresponding attribute does not exist in the resource advertisement, the attribute of the expression is replaced with the constant `undefined`. In our matchmaking algorithm, expressions containing `undefined` are eventually evaluated as *false*. The constraints of the resource advertisement have the similar evaluation process.

For example, assume that a consultation booking service needs at least 128K bytes free memory to run the service (i.e., `memoryfree`>=128K). The matchmaker scans the advertisement of the computing resource for the attribute `memoryfree`. The value of the attribute (e.g., 512000K bytes) is used to replace the attribute in the expression (i.e., 512000 >= 128), which is in turn evaluated to *true* by the matchmaker.

When receiving a request from a service migrator, the matchmaker takes the advertisement, evaluates all the resources advertised in the resource repository using the matchmaking algorithm described above, and returns a set of matched resources to the service migrator. To express it more precisely, we present a piece of pseudo code of the algorithm in Figure 3.

3.2 Resources Selection

For a specific service, there could be multiple computing resources matched for executing the service. The service provider, therefore, should be able to choose the best resource (or top N best resources) that satisfies her particular needs from the matched resources.

To specify preferences over resources of a particular Web service, we exploit a *multi-criteria utility function*,

$$\mathcal{U}(r) = \sum_{i \in \mathcal{SA}} w_i \cdot \mathcal{S}core_i(r)$$

where i) r is a resource, ii) $\mathcal{S}core_i(r)$ is an attribute scoring function, iii) $\mathcal{SA}$ is the set of selection attributes, and iv) w_i is the weight assigned to attribute i. The scoring service computes the weighted sum of criteria scores using the weight property of each selection criterion. It selects the resource of a service that produces the higher overall score according to the multi-criteria utility function. Several criteria—such as *price*, *availability*, *reliability*, and *reputation*—can be used in the function.

By using multi-criteria selection function, the best computing resource can be selected from the matched resources, which will be contacted by the service migrator for the consequent operations (e.g., migrate service code and invoke the service at the resource).

```
doMatchMaking(serviceDescription){
    SET matchedResources to empty
    FOR every advertised resource in Repository DO {
        matchR=matchRequirements(serviceRequirements,
            resourceDescription)
        matchC=matchConstraints(resourceConstraints,
            serviceDescription)
        IF (matchR and matchC)
            addResource(matchedResources, resource)
    }
    RETURN matchedResources
}

matchRequirements($\mathcal{SR}$, $\mathcal{RD}$){
    SET matchRequirements to false
    FOR every requirement expression in $\mathcal{SR}$ DO {
        IF evaluation(requirement)
            SET matchRequirements to true
        ELSE
           SET matchReuirements to false
            BREAK
    }
    RETURN matchRequirements
}

matchConstraints($\mathcal{RC}$, $\mathcal{SD}$){
    SET matchConstraints to false
    FOR every constraint expression in $\mathcal{RC}$ DO {
        IF evaluation(constraint)
            SET matchConstraints to true
        ELSE
            SET matchConstraints to false
            BREAK
    }
    RETURN matchConstraints
}
```

Figure 3: Pseudocode for resource matchmaking.

A service migrator may also select a number of matched resources (e.g., top N best resources) to form a *pool of service hosts* for that service. If the server where the service is currently running is heavily loaded, the migrator generates a new instance of the service (i.e., migrate a copy of the service and invoke it) on a service host with low workload in the pool. Obviously, multiple service hosts of a service can significantly increase the service availability.

4 RESOURCE SELECTION FOR COMPOSITE SERVICES

So far, we only consider the resource selection for an individual service where the selection is determined solely on the requirements of the service. Although this approach is sound enough to improve the availability of the individual (component) service (e.g., move the service to a new resource for the invocation in case the current server is overloaded), the *overall performance* of a composite Web service may not be optimized. For example, component services in a composite service may choose various resources. From a particular component service point of view, the performance is the best. Whereas from the composite service point of view, the performance might not be the best because these component services could all be put in a single service host or multiple service hosts of the same domain where the communication cost can be dramatically reduced.

In this section, we first introduce the concept of *execution plan* and then present an approach for selecting optimal execution plans for composite services.

4.1 Execution Plans

As stated before, the components of a composite service can be executed using several resources. Therefore, there can be multiple *execution plans* for a composite service. By execution plan, we mean those resources which can be used to execute a composite service. In other words, an execution plan indicates which resource is used for each component of a composite service.

Assume a composite service $\mathcal{CS}$ has m component services, $\mathcal{CS}_c = \{s_1, s_2, \cdots, s_m\}$, an execution plan p of $\mathcal{CS}$ is defined as follows:

Definition 1 (*Execution plan*). p = {<s_1, r_1>, <s_2, r_2>, $\cdots$, <s_m, r_m>} is an execution plan of composite service $\mathcal{S}$ if:

- $\bigcup_{i=1}^{m} s_i = \mathcal{CS}_c$, and
- for each 2-tuple <s_i, r_i> ($i \in [1,m]$) in p, the service s_i is executed in resource r_i. □

4.2 A Multi-phase Execution Planning

Building an optimal execution plan consists of several phases. In the following, we will describe and illustrate the phases using an example.

Phase 1: Matching Resources. The purpose of this phase is to search for the resources on which the component services could be executed. It should be noted that only those component services that are movable, their codes can be transferred to other computing resources, are involved in this stage [4].

The matchmaker evaluates the requirement and constraint expressions of the services and resources.

[4]The component services that can not move (i.e., stationary services) have only one "matched" resource (i.e., their own service servers).

If all the requirements of a component service s_i are evaluated to *true* using the attributes of a resource r_i, and all the constraint expressions of the resource r_i are evaluated to *true* using the attributes of the service s_i, we say r_i is a *matched resource* of s_i. A reference to a non-existent attribute evaluates to the constant `undefined` which is treated as *false*. Since there could be multiple matched resources of a specific service, for each component service s_i of a composite service $\mathcal{CS}$, the result of the matching phase of $\mathcal{CS}$ is represented as: $\mathcal{M}=\{<s_1, \mathcal{R}_1>, <s_2, \mathcal{R}_2>, \cdots, <s_m, \mathcal{R}_m>\}$, where $\mathcal{R}_i$ is the set of matched resources of service $s_i \in \mathcal{CS}_c$, represented as $\mathcal{R}_i=\{r_{i1}, r_{i2}, \cdots, r_{ik}\}$.

For example, imagine a composite service consists of 8 component services (s_1, s_2, $\cdots$, s_8) and the matching results of the composite service are given in Figure 4 (a). From the table we can see that resources r_1, r_2 and r_3 can be used to execute service s_1. The resource r_1 can also be used to execute service s_2, s_4, s_6, and s_8. Figure 4 (b) gives the domain information of these matched resources. For instance, resources r_1, r_2, r_3, and r_4 belong to a domain named `K-17`.

Phase 2: Pruning Phase. For component services which are supposed to run concurrently, if a resource r is shared by several such services, a *rematch procedure* must be performed to ensure that these component services *can* be executed in r concurrently.

For example, suppose s_2 and s_3 are executed in parallel and require at least 150K bytes and 120K bytes of free memory to carry out their executions, respectively. Also suppose that resource r_4 has 260K bytes of free memory. Intuitively, r_4 meets the requirements of both s_2 and s_3 separately. However, r_4 does not meet the requirements when s_2 and s_3 are executed at the same time because they need 270K bytes of free memory totally. As a result, the resource r_4 is removed from the sets of the matched resource of s_2 and s_4.

Phase 3: Generating Execution Plan. Since each component service s_i ($s_i \in \mathcal{CS}_c$) has a set of matched resources ($\mathcal{R}_i$), the association of a component service with a specific resource must be completed in order to build an execution plan for composite service $\mathcal{CS}$.

Initially, the work starts with the first component service s_i (i=1) of the composite service $\mathcal{CS}$. In this step, the best resource will be selected from $\mathcal{R}_i$ using the multi-attribute (e.g., price, availability) utility function (see Section 3.2). End users can customize the weights for the selection criteria in order to find a desired resource for the component service. For example, if the price is the most important factor to a customer, she can set its weight to 1. For each resource, a score is computed using the above utility function and the resource with the maximal value is selected. If there is more than one resource which have the same maximal value, then a resource will be chosen randomly from them. Suppose that resource r_i is finally selected to execute service s_i, the plan for executing service s_i is represented as $<s_i, r_i>$.

After the preparation of the first component service s_i (i=1) is finished, the resources should be selected for the remaining component services s_i (i=2 $\ldots$ m). The location criterion is considered at this stage. Obviously, the resource that is going to be assigned to a component service s_i (i=2 $\ldots$ m) depends on the location of the resource that is assigned to its predecessor s_{i-1}. In particular, if the resources $\mathcal{R}_i$ of service s_i contains the resource selected for s_{i-1} (i.e., r_{i-1}), the resource should also be selected for s_i. Otherwise, a number of resources $\mathcal{R}_{domain}$ that have the same domain as r_{i-1} are selected from $\mathcal{R}_i$. The best resource is selected from the $\mathcal{R}_{domain}$ by using the utility function in Step 1. If there is no resource having the same domain with r_{i-1}, a resource will be selected using the utility function by going to Step 1. For example, suppose that r_9 is selected for s_6, a resource will be selected from r_{10} and r_{11} for s_7 because r_9, r_{10}, and r_{11} has the same domain (see Figure 4 (b)).

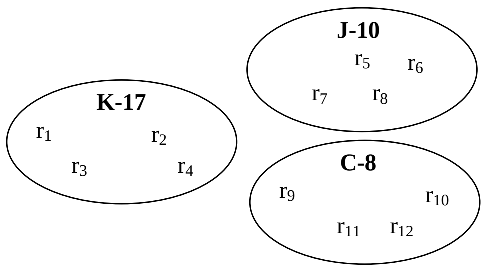
After the optimized execution plan of $\mathcal{CS}$ is built, when the composite service $\mathcal{CS}$ is invoked, the migrator of its component services are in charge of migrating and invoking the services in the assigned computing resources.

5 DISCUSSIONS

The work presented in this paper is related to robust provisioning of Web services. Regarding the reliability and availability of Web services, several proposals have been presented in the literature (Casati and Shan, 2001; Keidl et al., 2003). The eFlow system (Casati and Shan, 2001) provides a dynamic service selection technique. With dynamic service selection, a composite service searches for component services based on available metadata, its own internal state, and a rating function. In contrast, our work allows Web services to be highly available by moving the execution of services to other service hosts dynamically. The *dispatcher service* that is capable of automatic service replication presented in (Keidl et al., 2003) is most similar to our work. However, the strategies on service hosts selection have not been discussed. Instead, a dispatcher is responsible for a set of service hosts that are known a priori. In addition, their approach only considers individual services, not composite services. Comparing to their work, we not only present

Service	Matched resources
s_1	$\{r_1, r_2, r_3\}$
s_2	$\{r_1, r_4\}$
s_3	$\{r_4, r_5, r_6\}$
s_4	$\{r_1, r_7, r_8\}$
s_5	$\{r_4, r_8, r_9\}$
s_6	$\{r_1, r_9\}$
s_7	$\{r_3, r_7, r_{10}, r_{11}\}$
s_8	$\{r_1, r_{10}, r_8, r_{12}\}$

(a)



(b)

Figure 4: (a) Matched resources of the composite service; (b) domains of the matched resources.

an approach for resources matchmaking where appropriate service hosts are selected dynamically, but also present a multi-phase execution planning mechanism for the high availability and best overall performance of composite Web services. It is also worth mentioning that the Web Service Reliability (WS-Reliability) specification (Iwasa et. al.,) is an industry effort for open, reliable Web services messaging including guaranteed delivery, duplicate message elimination and message ordering. However, although the specification was named as "Web service reliability", it deals less with the overall reliability of Web services than with "Web services reliable message delivery".

With regard to the matchmaking approaches, InfoSleuth (Nodine et al., 2003) is an agent-based information discovery and retrieval system that adopts *broker agents* to perform the syntactic and semantic matchmaking. The broker agent matches agents that require services with the agents that can provide such services. In InfoSleuth, the service capabilities is written in LDL++ (Chimenti et al., 1990), a logical deduction language. RETSINA (Sycara et al., 1999) is a multiagent system for dynamic service matchmaking on the Web. The authors distinguished three kinds of agents in Cyberspace: *service provider*, *service requester*, and *middle agent*. An appropriate service provider is matched to a requester through the middle agent. A language is designed for the description of agents capabilities. In contrast, our matchmaking between services and resources is based on an approach where providers of services and resources can express their constraints and requirements. In addition, we propose a multi-criteria utility function for selecting optimal resources for particular services.

The techniques presented in this paper have been implemented in Self-Serv system (Sheng, 2006) as an extension to Web services development. In particular, UDDI is used as the service and resource repository. We defined an XML schema using RDF for resource description. The HP Jena Toolkit 1.6.1[5] is used to manipulate RDF documents. WSDL is used to specify Web services. Since WSDL focuses on how to invoke a Web service, some of the attributes (e.g., stationary or mobile service) in our approach are not supported. To overcome this limitation, such attributes are specified as tModels. The keys of these tModels are included into the `categoryBag` of the tModel of a Web service. Finally, μCode 1.03[6] is used to implement mobile agents for service invocation at service hosts as well as results collection.

Our ongoing work includes the performance study of the proposed techniques on service migration. We are also planning to examine the support of dynamic information of resources and services that changes over the time such as amount of free memory.

REFERENCES

Acharya, A., Ranganathan, M., and Saltz, J. (1997). Sumatra: A Language for Resource-Aware Mobile Programs. *Mobile Object Systems: Towards the Programmable Internet*, pages 111–130.

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2004). *Web Services Concepts, Architectures and Applications*. Springer Verlag.

Benatallah, B., Sheng, Q. Z., and Dumas, M. (2003). The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1):40–48.

Casati, F. and Shan, M.-C. (2001). Dynamic and Adaptive Composition of E-Services. *Information Systems*, 26(3):143–162.

Chen, Y.-F. R. and Petrie, C. (2003). Ubiquitous Mobile Computing. *IEEE Internet Computing*, 7(2):16–17.

Chimenti, D., Gamboa, R., Krishnamurthy, R., Naqvi, S. A., Tsur, S., and Zaniolo, C. (1990). The LDL System Prototype. *IEEE Transactions on Software Engineering*, 2(1):76–90.

Curbera, F., Khalaf, R., Mukhi, N., Tai, S., and Weerawarana, S. (2003). The Next Step in Web Services. *Communications of The ACM*, 46(10):29–34.

[5]http://jena.sourceforge.net/.

[6]http://mucode.sourceforge.net/.

Fuggetta, A., Picco, G. P., and Vigna, G. (1998). Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361.

Ingham, D. B., Shrivastava, S. K., and Panzieri, F. (2000). Constructing Dependable Web Services. *IEEE Internet Computing*, 4(1):25–33.

Ishikawa, F., Yoshioka, N., Tahara, Y., and Honiden, S. (2004). Toward Synthesis of Web Services and Mobile Agents. In *Proc. of AAMAS'2004 Workshop on Web Services and Agent-based Engineering (WS-ABE2004)*, New York, USA.

Iwasa et. al., K. Web Service Reliability Specification. `http://www.oasis-open.org/specs/index.php`.

Keidl, M., Seltzsam, S., and Kemper, A. (2003). Reliable Web Service Execution and Deployment in Dynamic Environments. In *Proc. of the 4th VLDB Workshop on Technologies for E-Services (VLDB-TES03)*, Berlin, Germany.

Liu, P. and Lewis, M. J. (2005). Mobile Code Enabled Web Services. In *Proc. of IEEE International Conference on Web Services (ICWS'05)*, Orlando FL, USA.

Nodine, M. H., Ngu, A. H. H., Cassandra, A. R., and Bohrer, W. (2003). Scalable Semantic Brokering over Dynamic Heterogeneous Data Sources in InfoSleuth$^{\text{TM}}$. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1082–1098.

Sheng, Q. Z. (2006). *Composite Web Services Provisioning in Dynamic Environments*. PhD thesis, The University of New South Wales, Sydney, NSW, Australia.

Sycara, K., Klusch, M., Widoff, S., and Lu, J. (1999). Dynamic Service Matchmaking Among Agents in Open Information Environments. *ACM SIGMOD Record*, 28(1):47–53.